(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 037 148 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
20.09.2000 Bulletin 2000/38

(51) Int. Cl.^{7}: G06F 11/10

(21) Application number: 00200916.5

(22) Date of filing: 14.03.2000

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 15.03.1999 US 124483 P

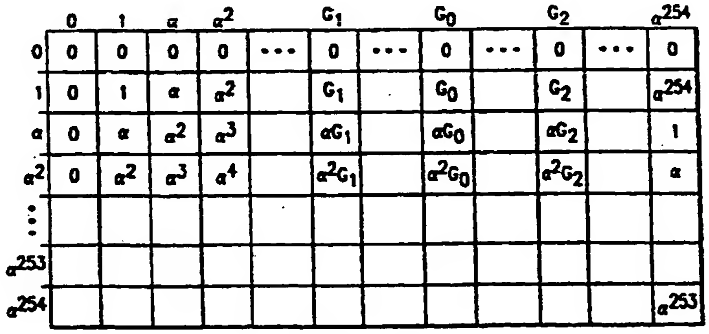
(71) Applicant:
Texas Instruments Incorporated
Dallas, Texas 75251 (US)

(72) Inventors:
- Cheng, Yaqi
Smyrna, Georgia 30080 (US)
- Polley, Michael O.
Richardson, Texas 75082 (US)

(74) Representative: Holt, Michael
Texas Instruments Limited,
European Patents Department (MS 13),
PO Box 5069
Northampton NN4 7ZE (GB)

(54) Error coding method

(57) A decimated and interleaved multiplication table for finite fields as is useful in Reed-Solomon encoding computations. The generator polynomial coefficients determine the multiplication table content and ordering.



	0	1	α	α^2		G_1		G_0		G_2		α^{254}
0	0	0	0	0	•••	0	•••	0	•••	0	•••	0
1	0	1	α	α^2		G_1		G_0		G_2		α^{254}
α	0	α	α^2	α^3		αG_1		αG_0		αG_2		1
α^2	0	α^2	α^3	α^4		$\alpha^2 G_1$		$\alpha^2 G_0$		$\alpha^2 G_2$		α
⋮												
α^{253}												
α^{254}												α^{253}

FIG. 5

Printed by Xerox (UK) Business Services
2.16.7 (HRS)/3.6

Description

FIELD OF THE INVENTION

[0001] The present invention relates generally to the technical field of electronic devices, and, more particularly, to error correction coding in such electronic devices.

BACKGROUND TO THE INVENTION

[0002] Digital communication and storage systems typically include error correction coding in order to overcome errors arising from the transmission or storage medium. Forward error-correction coding (FEC) systems add redundancy to the transmitted signal so that the receiver can detect and correct errors using only the received signal. This eliminates the need for the receiver to send requests for retransmission to the transmitter.

[0003] One of the more popular error correction code types is Reed-Solomon code. Reed-Solomon codes are block codes with maximum distance separation and are highly efficient in their use of redundancy. The most appealing aspect of Reed-Solomon codes is the availability of efficient decoding algorithms. See for example, Wicker and Bhargava (Eds.) Reed-Solomon Codes and Their Applications (IEEE Press, Piscataway, New Jersey, 1994).

[0004] Figure 1 shows a high level view of a Reed-Solomon coding system. A block of information bits, I, is encoded into a codeword C which is a larger block that contains the original information and redundant bits. After transmission over a channel, the received block of bits can be represented as C+E where E is a block of error bits. The decoder generates I' from C+E, and I' will equal I if the number of errors bits in E is within the correction capabilities of the code.

[0005] Figure 2 shows a more detailed description of Reed-Solomon coding. In particular, a block of bk information bits is divided into k groups of b bits and each group of b bits is represented by a symbol, producing a block of k information symbols for coding. The encoder operates on the block of k information symbols to produce a block of n codeword symbols containing the original information in some form as well as redundancy. The code can be designed so that the redundancy is used for error detection only, error correction only, or a combination of some error detection and some error correction. The block of n coded symbols is then translated into a block of bn bits and transmitted over the channel. The receiver front-end produces a block of bn bits that might be corrupted, depending upon the amount of channel distortion. The block of bn bits is translated into a block of n symbols and processed with the decoder. As long as the transmission errors lead to at most (n-k)/2 erroneous symbols, a hard-decision decoder can reliably recover the input k information symbols and input bk bits. The price paid for the added redundancy is the increase in the number of symbols to transmit by a factor of n/k. Of course, this means an information decrease by a factor of k/n for a constant transmission rate.

[0006] Reed-Solomon encoding essentially maps k information symbols with the symbols as elements of a finite field (Galois field or GF) with a power of 2 number of elements into n symbols which are GF elements from the sane finite field to form a codeword. Note that for such field with 2^M elements, denoted $GF(2^M)$, the elements can be represented by M-bit words and the nonzero elements can be expressed as powers of a primitive element α. That is, the elements of $GF(2^M)$ are 0, 1, α, α^2, ..., α^q where $q = 2^M - 2$.

[0007] Nonsystematic Reed-Solomon encoding produces codewords by distributing the information and redundancy across the entire codeword of n symbols according to the coding algorithm. Systematic Reed-Solomon encoding, on the other hand, forms codewords by concatenating the k information symbols with n-k parity symbols, which are computed according to the coding algorithms. The additional n-k parity symbols contain the redundant information that is used by the receiver to choose the most likely transmitted k information symbols. In particular, with receiver soft decision the n-k parity symbols can be used to correct e error symbols and detect s erased symbols provided 2e+s is at most equal to n-k. Note that values such as n = 204 and k = 188 with the GF being $GF(2^8)$ (the finite field with 256 elements) are not uncommon. Indeed, this is a commonly used code for high speed modems and would be called a (204,188) code. This code can correct 8 error symbols per 204-symbol codeword.

[0008] Systematic Reed-Solomon encoding is advantageous because the information component of the received codeword can be extracted at the receiver without applying the Reed-Solomon decoding operations. The first k symbols represent all of the information. The last n-k symbols must be computed from the information symbols.

[0009] The parity symbols can be computed from the information symbols using methods based on the arithmetic of polynomials whose coefficients are GF elements with the elements representing groups of bits. The information, parity, and codewords are represented by polynomials I(x), P(x), and C(x), respectively. For systematic Reed-Solomon encoding $C(x) = x^{n-k}I(x) + P(x)$ with P(x) the remainder from the polynomial division of $x^{n-k}I(x)$ by G(x). G(x) is the generator polynomial of the code and is a monic polynomial of degree n-k: $G(x) = x^{n-k} + G_{n-k-1}x^{n-k-1} + G_{n-k-2}x^{n-k-2} + \ldots + G_1x + G_0$, so P(x) has degree at most n-k-1. I(x) is a polynomial of degree at most k-1 with the k coefficients being the k information symbols, so C(x) is a polynomial of degree at most n-1 with coefficients being the n codeword symbols.

[0010] Most popular architectures that implement polynomial division for systematic Reed-Solomon encoding comprise feedback shift registers that are composed of delay elements, GF element multipliers, and GF element adders as shown in Figure 3 The delay elements D are initialized with zero symbol values. The information symbols are shifted into the register one at a time, highest order element first (I_{k-1}). During each clock cycle of the register, the GF element held in the last delay element (leftmost) is fed back to n-k multipliers that compute the product of the feedback element with the feedback register multiplier elements G_0 through G_{n-k-1}. Because the finite field has a power of 2 number of elements, subtraction and addition are the same operation.

[0011] At each stage of the feedback register the products are added to the stored elements in the previous stage and the result is stored in the following stage. After clocking the register n times the elements stored in the delay elements D are the remainder of the division, or the parity elements that constitute the coefficients of the parity polynomial P(x). Figure 4 shows a simplified feedback shift register that uses a pre-shifted I(x) to compute the remainder in only n-k clock cycles of the register.

[0012] It is important to understand that he architectures shown in Figures 3-4 evolved from a desire to efficiently implement Reed-Solomon encoders with circuit elements. In a typical encoder design, the three types of circuit elements (delays, GF adders, and GF multipliers) are individually optimized and then put together to perform the desired remainder computation operation. This type of encoder architecture can be emulated on a general purpose digital signal processing (DSP) platform. However, while either the GF multiply or the GF add can be implemented efficiently (depending upon the particular GF representation used), they cannot both be implemented efficiently simultaneously. For example, one particular representation allows GF adds to be computed with a simple exclusive-OR operation of the binary components of the two elements. In general, this can be implemented in one cycle of a DSP. However, for this same GF element representation, the GF multiply requires a large number of cycles to compute.

[0013] A GF multiplication table can be employed to reduce the number of cycles required to multiply two GF elements. However, a GF multiplication table can require a large amount of memory, and memory lookup to determine the product of two GF elements can also be somewhat time consuming.

SUMMARY OF THE INVENTION

[0014] The present application discloses a simplified finite field (Galois field or GF) multiplication table which is decimated and interleaved.

[0015] The present application further discloses a method of performing polynomial division with finite field coefficients. The method provides a finite field multiplication table with entries consisting of multiples of the coefficients of a divisor polynomial and ordered according to the coefficients of said divisor polynomial. Partial quotients and reminders are then iteratively computed.

[0016] This has the advantages of allowing general purpose digital signal processors (DSPs) to efficiently perform Reed-Solomon encoding and thereby eliminates the need for specialized feedback shift register circuitry.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] Specific embodiments of the present invention will now be described in further details, by way of example, with reference to the accompanying drawings in which;

Figures 1-2 schematically illustrate Reed-Solomon coding.
Figures 3-4 show shift registres for polynomial division.
Figure 5 illustrates a GF(256) multiplication table; and
Figure 6 shows a preferred embodiment GF(256) multiplication table.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0018] Figures 5-6 illustrate the preferred embodiment GF(2^8) element multiplication lookup table construction by selection (decimation) of columns from a full multiplication lookup table and reordering (interleaving) for efficient access. The coefficients of the Reed-Solomon code generator polynomial as elements of the GF indicate the columns to be retained, and the memory locations used for the emulation of the delays sets the ordering of the columns. Note for a (204,188) code, the code generator polynomial G(x) has degree 16, the information polynomial I(x) has degree 187, and the parity polynomial P(x) has degree 15; so the polynomial division has 188 steps with each step requiring 16 multiplications and 16 additions. GF coefficient polynomial division

[0019] The preferred embodiments emulate the feedback shift register of Figure 4 with a general purpose processor; thus first consider the shift register operation. The polynomial division of $x^{n-k}I(x)$ by G(x) to yield P(x) proceeds in steps with each step adding the next lower power of x term to the quotient and leaving a remainder of one lower power of x. The last step generates the constant term of the quotient and yields the coefficients of the final remainder, P(x), in the delays. Each step corresponds to one clock cycle in the feedback register of Figure 4. Now:

$$G(x) = x^{n-k} + G_{n-k-1}x^{n-k-1} + \dots + G_0 \text{ and } I(x) = I_{k-1}x^{k-1} + I_{k-2}x^{k-2} + I_{k-3}x^{k-3} + \dots + I_{0.}$$

Thus:

$$x^{n-k}I(x) = I_{k-1}x^{n-1} + I_{k-2}x^{n-2} + I_{k-3}x^{n-3} + \dots + I_0x^{n-k},$$

And the first step of the division gives a first quotient term $I_{k-1}x^{k-1}$ and a first remainder equal to:

$$(I_{k-2}-G_{n-k-1}I_{k-1})x^{n-2} + (I_{k-3}-G_{n-k-2}I_{k-1})x^{n-3} + \dots + (I_{k-1-(n-k)}-G_0I_{k-1})x^{k-1} \\ + I_{k-2-(n-k)}x^{k-2} + I_{k-3-(n-k)}x^{k-3} + \dots + I_0x^{n-k},$$

During the first clock cycle the feedback shift register of Figure 4 shifts in I_{k-1}, computes the products G_jI_{k-1} for j = 0 to n-k-1 , and stores them in the delays 0, 1, ..., n-k-1 . The first clock shifted in I_{k-1} but the terms I_{k-2}, I_{k-3}, ... , I_0, have not yet been shifted in. The second step then gives a second quotient term:

$$(I_{k-2}-G_{n-k-1}I_{k-1})x^{k-2}$$

And a second remainder

$$(I_{k-3}-G_{n-k-2}I_{k-1}-(I_{k-2}-G_{n-k-1}I_{k-1})G_{n-k-1})x^{n-3} + (I_{k-4}-G_{n-k-3}I_{k-1}-(I_{k-2}-G_{n-k-1}I_{k-1}) \\ G_{n-k-2})x^{n-4} + \dots + (I_{k-1-(n-k)}-G_0I_{k-1} - (I_{k-2}-G_{n-k-1}I_{k-1})G_1)x^{k-1} + (I_{k-2-(n-k)} - \\ (I_{k-2}-G_{n-k-1}I_{k-1})G_0)x^{k-2} + I_{k-3-(n-k)}x^{k-3} + \dots + I_0x^{n-k},$$

During the second clock cycle the shift register shifts in I_{k-2}, computes the terms $(I_{k-2}-G_{n-k-1}I_{k-1})G_j$ for j = 0, ... , n-k-1 , subtracts (adds) them (except for j = 0) from the terms $G_{j-1}I_{k-1}$, and stores in the delays. Note that the terms $(I_{k-2}-G_{n-k-1}I_{k-1})G_j$, are computed by first subtracting (adding) the product $G_{n-k-1}I_{k-1}$ (stored in delay n-k-1 by the previous clock cycle) from the shifted in I_{k-2} and then multiplying the result by G_j in each of the multipliers. Also, the $G_{j-1}I_{k-1}$ term to add to this product were stored in the adjacent delay from the first clock cycle and is shifted in for the addition and storage. Again, the terms I_{k-3}, ... , I_0 have not yet been shifted in.

[0020] Similarly, the successive division steps build up the remainders until the kth and last step has the remainder P(x) coefficients P_{n-k-1}, ... , P_0 in the delays n-k-1 , ..., 0.

[0021] The preferred embodiments rely on the following analysis of the foregoing polynomial division. On the ith step the symbol, M_{n-k-1}, stored in the n-k-1 memory is subtracted (added) from the symbol I_{k-I} shifted in, and the result is multiplied by each of the generator polynomial coefficients, G_{n-k-1}, ..., G_0. Thus if the symbol $I_{k-I} - M_{n-k-1}$ is denoted α^m where α is the primitive element of the GF, then the n-k multiplications are $\alpha^m G_{n-k-1}$, $\alpha^m G_{n-k-2}$, ..., and $\alpha^m G_0$. Hence, the multiplicands are all the same, namely, α^m.

[0022] Now, consider implementing these multiplcations by a lookup table. The element α^m is used to index one of the rows of the multiplication table and the element G_j indexes one of the columns of the table. The consecutive multiplications executed during the clock cycle will require the same row to be accessed each time, the columns indexed correspond to elements G_{n-k-1}, ..., G_0. Therefore, the products produced by the consecutive multiplies are extracted from a given row and various columns of the table. In general, the column indices will not be consecutively ordered. For any clock cycle of the register the row index can change, but the column indexing pattern will remain the same.

[0023] The amount of memory required to store the multiplication table can be reduced by eliminating the columns that are never indexed. The remaining columns in the multipication table correspond to the elements G_{n-k-1}, ..., G_0. Thus the memory size to store the new multiplication table with only these columns is 2^M by n-k where GF is the finite field $GF(2^M)$. For example, with $GF(2^8)$, the full table would be 256 by 256 elements, but with the (204,188) code the table would be only 256 by 16 elements.

[0024] The number of DSP cycles required to implement the GF multiplies by table lookup can be reduced by ordering the columns so that they correspond to the consecutive elements G_{n-k-1}, ..., G_0 (or G_0, G_1, ..., G_{n-k-1}, depending on the ordering of the memory locations emulating the delays). After ordering columns in this manner, multiplication by α^m

is implemented by accessing the appropriate row and then reading out the consecutive n-k elements of that row.

[0025] The decimated multiplication table implementation can be further simplified by collapsing it into a one-dimensional structure where the α^m element determines the offset into the one-dimensional memory array. Then, n-k consecutive elements are read from consecutive memory locations in the array. In the case where each element is smaller than the accessible memory element size, several products can be accessed simultaneously. For example, if each product is an 8-bit GF element, and the DSP can access 32-bit words from memory, then four products can be read in one memory access.

[0026] Figure 5 illustrates the conceptual manner in which the GF(2^8) products of interest are extracted from a full-sized GF multiplication table and Figure 6 shows them placed into the decimated and interleaved GF multiplication table. In this example, each GF element is represented by 8 bits. Therefore the full multiplication table contains 256 by 256 elements. The first row is trivially 0s, and so are the first n-k elements of the new array. Starting with the second row (for element $\alpha^0 = 1$), product element $\alpha^0 G_0$ is placed in the new array, element $\alpha^0 G_1$ is placed in the second location, and so forth up to element $\alpha^0 G_{n-k-1}$. Thus, the second n-k elements are the products of α^0 and the ordered elements (coefficients) of the generator polynomial G(x). The products of α^1 and the ordered elements of G(x) are placed in the next n-k array locations, and so forth. During operation, result of multiplying of α^j by the ordered elements of G(x) is obtained by accessing the n-k elements in the memory array starting at the index j(n-k).

[0027] Figure 6 illustrates the decimated and interleaved GF multiplication table of 256(n-k) elements. An 8-bit input or multiplicand is 0 or α^j for j=0,1,2,...,254. The shift register multipliers (the other multiplicands) are G_0, G_1, ..., G_{n-k-1}. The products are stored as an interleaved table so that all products for any input and the shift-register multipliers can be looked up in the appropriate order within one block. This makes it possible to perform multiple multiplications in one lookup cycle. For instance, a 32-bit load can produce the products for four 8-bit GF multiplications. This is highly efficient for DSP implementation with wide memory busses.

[0028] The preferred embodiments can be modified in various ways while retaining the feature of a multiplication table decimated to the Reed-Solomon code generator polynomial coefficients and interleaved for sequential access.

[0029] For example, the finite field may have a different size; the sequential access may be adapted to another method of polynomial division;

Claims

1. A method of performing polynomial division with finite field coefficients, which method comprising:

   providing a finite field multiplication table with entries consisting of multiples of the coefficients of a divisor polynomial and ordered according to said coefficients of said divisor polynomial; and
   iteratively computing partial quotients and remainders.

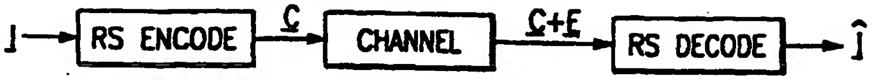

I
RS ENCODE
C
CHANNEL
C+E
RS DECODE
Î

FIG. 1

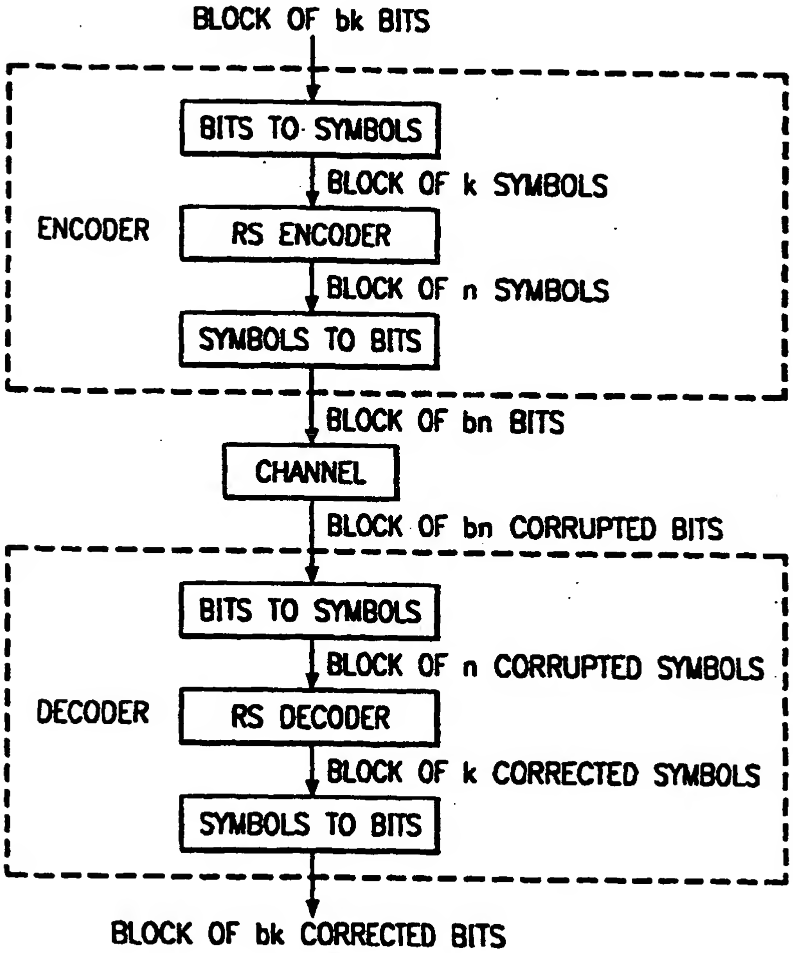

BLOCK OF bk BITS
ENCODER
BITS TO SYMBOLS
BLOCK OF k SYMBOLS
RS ENCODER
BLOCK OF n SYMBOLS
SYMBOLS TO BITS
BLOCK OF bn BITS
CHANNEL
BLOCK OF bn CORRUPTED BITS
DECODER
BITS TO SYMBOLS
BLOCK OF n CORRUPTED SYMBOLS
RS DECODER
BLOCK OF k CORRECTED SYMBOLS
SYMBOLS TO BITS
BLOCK OF bk CORRECTED BITS

FIG. 2

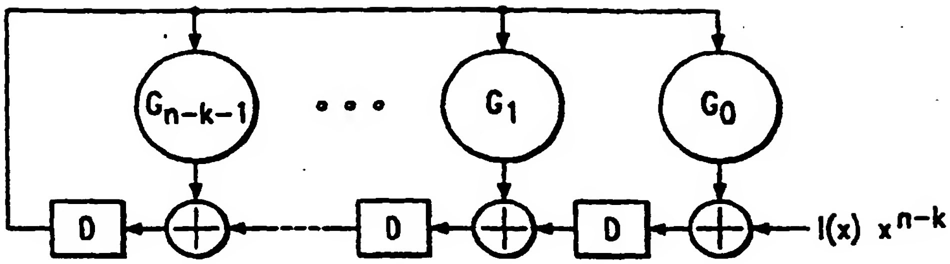


FIG. 3

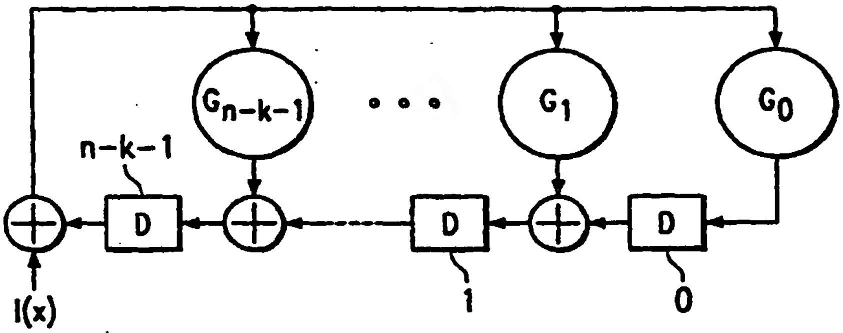


FIG. 4

	0	1	α	α^2		G_1		G_0		G_2		α^{254}
0	0	0	0	0	• • •	0	• • •	0	• • •	0	• • •	0
1	0	1	α	α^2		G_1		G_0		G_2		α^{254}
α	0	α	α^2	α^3		αG_1		αG_0		αG_2		1
α^2	0	α^2	α^3	α^4		$\alpha^2 G_1$		$\alpha^2 G_0$		$\alpha^2 G_2$		α
⋮												
α^{253}												
α^{254}												α^{253}

FIG. 5

INDEX	ELEMENT
0	0
1	0
	⋮
n−k−1	0
n−k	G_0
n−k+1	G_1
	⋮
2(n−k)−1	G_{n-k-1}
2(n−k)	αG_0
	αG_1
	⋮
3(n−k)−1	αG_{n-k-1}
3(n−k)	$\alpha^2 G_0$
	$\alpha^2 G_1$
	⋮
	$\alpha^2 G_{n-k-1}$
4(n−k)	$\alpha^3 G_0$
	⋮
256(n−k)−1	$\alpha^{254} G_{n-k-1}$

FIG. 6

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 00200916.5

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl. 7)
X	US 4498175 A (NAGUMO et al.) 05 February 1985, the whole document. --	1	G06F11/10
A	WO 95/15033 A (THOMSON CONSUMER ELECTRONICS INC.) 01 June 1995, the whole document. --	1	
A	US 4937829 A (KADOKAWA) 26 June 1990, the whole document. ----	1	
			TECHNICAL FIELDS SEARCHED (Int. Cl. 7)
			G06F H03M

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
VIENNA	04-05-2000	STEPANOVSKY

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 00200916.5

This annex lists the patent family members relating to the patent documents cited in the above-mentioned search report. The members are as contained in the EPIDOS INPADOC file on 18.05.2000. The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US A 4498175	05-02-1985	DE C0 3278677	21-07-1988
		EP A2 96109	21-12-1983
		EP A3 96109	24-10-1984
		EP B1 96109	15-06-1988
		JP A2 58219852	21-12-1983
		JP B4 63008651	24-02-1988
		KR B1 8600J03	16-07-1986
WO A1 9515033	01-06-1995	AU A1 56746/94	13-06-1995
		BR A 9307901	27-08-1996
		CN A 1111858	15-11-1995
		EP A1 730795	11-09-1996
		EP A4 730795	04-02-1998
		JP T2 9505703	03-06-1997
		US A 5912907	15-06-1999
US A 4937829	26-06-1990	JP A2 1039832	10-02-1989
		JP B2 2622383	18-06-1997
		JP A2 63274221	11-11-1988
		JP A2 63267018	04-11-1988

For more details about this annex see Official Journal of the European Patent Office, No. 12/82.